VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Realizace síťové strategické hry

Realization of a Network Strategic Game

2010

Janošek Lumír

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2010 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Rád bych na tomto místě poděkoval Ing. Marku Běhálkovi Ph.D. , který mi s prací pomohl a bez kterého by tato práce nevznikla.

Abstrakt

Cílem této práce je položit základ budoucímu vývoji jednoduchého herního enginu zaměřeného převážně na hry typu strategie nebo RPG. V práci je prezentován současný stav vývoje a co je od projektu očekáváno do budoucna. Nastíněn je také koncept hry, která bude na tomto enginu v budoucnu postavena. Obsahově se práce zaměřuje výhradně na grafickou část, tedy vykreslování 3D grafiky, a jednoduchou optimalizaci scény. Je zde shrnuta architektura vyvíjeného enginu a popsána implementace jeho významnějších částí. Na vývoji enginu spolupracuji s Lubomírem Březinou, který se zabývá vývojem umělé inteligence a části enginu obsluhující síťovou hru.

Klíčová slova: engine, rendering, entita, zařízení Direct3D, scéna

Abstract

The aim of this thesis is to lay the foundations of the future development of the simple game engine focused on the strategy or RPG games mostly. In the thesis is presented the current state of the development and the expectations of the future. There is also outlined the concept of the game, which would be constructed on this engine. The content of the thesis is focused on the graphic section, the 3D depiction, and the simple optimalization of the scene. There is summarized the architecture of the developed engine and described the implementation of its important sections. I work on this engine with Lubomír Březina, who is occupied with the development of the artificial intelligence and with the sections of the engine which operate the network game.

Keywords: engine, rendering, entity, Direct3D device, scene

Seznam použitých zkratek a symbolů

FPS	–	Frames Per Second
CPU	–	Central Processing Unit
GPU	–	Graphic Processing Unit
HLSL	–	High Level Shader Language
AABB	–	Axis Aligned Bounding Box
OOBB	–	Object Oriented Bounding Box

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Téma této bakalářské práce jsem si vybral z důvodu mého zájmu o tvorbu počítačových her, především jejich grafické stránky. Jedním z cílů této práce je vytvořit základ pro budoucí vývoj enginu pro konkrétní počítačovou hru. Pracovně byla tato hra pojmenovaná *Heroes*. Přestože je v současné době dostupných nespočet hotových herních enginů, ať už komerčních nebo freewarových, jejichž stručný přehled některých zástupců následuje níže, pokládal jsem za vhodnější pokusit se vytvořit vlastní. Né však z důvodu překonání kvalit dnes dostupných enginů, ale především pro lepší proniknutí do dané problematiky.

Na této práci spolupracuji v týmu s Lubomírem Březinou, který pracuje na algoritmech umělé inteligenci a síťové části enginu. Mým cílem v této práci je vytvořit základ renderovací části enginu se správou scény a entit, tedy objektů, které tuto scénu utvářejí. V budoucnu je očekáván větší rozvoj tohoto enginu a případné doplnění o další části, jako je například práce se zvukem.

Pro hardwarově urychlený rendering počítačové grafiky jsou dnes dostupné dvě nejrozšířenější rozhraní Direct3D, které je součástí většího balíku aplikačních programových rozhraní DirectX dodávaného společností Microsoft, a průmyslově standardizované rozhraní OpenGL (Open Graphic Library). V této práci jsem zvolil pro práci s 3D grafikou rozhraní Direct3D, protože je primárně určeno pro vývoj grafiky v počítačových hrách.

Na začátku této práce je uveden stručný přehled a charakteristika alternativních technologií a použitých technologií.

V kapitole 4 je uvedena bližší specifikace zadání, tedy co je od enginu a hry očekáváno, a jsou zde nastíněny plány budoucího rozvoje projektu. Je zde popsán také základní koncept samotné hry.

V kapitole 5 je uveden návrh implementace enginu a jeho architektura. Z důvodu nedostatku místa ovšem není vše rozvedeno do detailu. Zmíněny jsou i plány, jakým by se měl vývoj projektu ubírat v budoucnu.

V kapitole 6 je uveden základní koncept použitý při implementaci reprezentace scény. Je zde nastíněn způsob, jakým je implementováno dělení prostoru a reprezentace entit.

V kapitole 7 jsou na začátku popsány základní optimalizační algoritmy implementované grafickou kartou. Dále je pak popsán způsob implementace vylučování objektů na základě pohledového jehlanu, použitý v této práci.

V kapitole 8 je popsána vlastní implementace způsobu reprezentace scény za použití QuadTree. Je zde popsána metoda efektivního vkládání do této struktury a způsob implementace v enginu.

2 Alternativní technologie

Pod pojmem „herní engine“[1] se už v dnešní době nemyslí jen základ na kterém je vystavěna grafická stránka hry, i když toto je zřejmě stále hlavním aspektem. Herní engin se ve většině případů stará ve větší míře i o zvuk, animaci charakterů, fyzikální stránku, síťovou komunikaci, umělou inteligenci AI a v některých případech i o ochranu proti kopírování. Pojem „herní engine“ by se dal definovat jako komplexní softwarový systém navržený pro vytváření a vývoj počítačových her. Kromě těchto základních vlastností by měl herní engine zvládat také správu scény a objektů (entit) ve scéně. Na starost by také měl mít zajišťování optimalizovaného vstupu entit do rendereru, za použití technik pro optimalizaci scény.

V dnešní době je dostupný nespočet herních enginů, jak freewarových tak i komerčních. V této kapitole se nesnažím vytvořit kompletní přehled všech známých dostupných enginů, ale jen stručné shrnutí známějších zástupců z nekomerční i komerční sféry, a přehled jejich základních vlastností.

2.1 Nekomerční enginy

Nekomerční enginy se ve většině případů nedají s komerčními srovnávat. Nekomerční enginy nabízejí řešení grafické stránky věci, ale ostatní vlastnosti, pro vývoj hry neméně důležité, jako je již zmíněná fyzika, zvuk, umělá inteligence a další, již neobsahují nebo jsou řešený pomocí knihoven a SDK z třetí ruky. Pro fyziku je například dostupná velmi kvalitní multiplatformní knihovna Newton Game Dynamics, kterou lze integrovat do enginu. Pro implementaci zvukové stránky je zase například dostupná multiplatformní knihovna OpenAL (Open Audio Library), vyvinutá společností Loki Software a dnes udržovaná Creative Technology za podpory Applu a různých nadšenců.

2.1.1 Irrlicht

Irrlicht[7] engine je multiplatformní vysoce výkonný free open source 3D engine napsaný v C++. Je vydáván pod licencí založené na zlib/libpng. Nejedná se čistě o herní engine, ale pouze o grafický engine. Podporuje širokou škálu renderovacích API, což zajišťuje onu platformní nezávislost. Kromě Direct3D 8.1, Direct3D 9.0 a OpenGL jsou přítomny i softwarové renderery. Aplikace využívající Irrlicht engine je možné spustit na platformě Windows, Linux, Mac OSX, Sun Solaris/SPARC a na všech platformách využívající SDL (Simple DirectMedia Layer), což je multiplatformní multimediální knihovna poskytující nízkoúrovňový přístup ke zvukovému hardwaru, klávesnici, myši, joysticku, 2D video framebufferu a 3D hardwaru skrze rozhraní OpenGL. Tento engine najde využití nejen pro vývoj realtime aplikací, jako jsou počítačové hry, ale i pro vědecké vizualizace.

Scéna je reprezentována za užití hierarchického scénového grafu, kde jednotlivými uzly mohou být, kromě běžných objektů scény, i indoor nebo outdoor části. Což dovoluje enginu míchat indoorové a outdoorové scény plynule dohromady. Je zde široké podpora vestavěných materiálů založených nejen na pevné funkční pipeline, ale i na programovatelné funkční pipeline. To dovoluje vytvoření například per pixel nasvětlovaných ma-

teriálů za užití normálového bump-mappingu nebo parallax mappingu. Vestavěná je také podpora animovaných modelů s využitím skeletální animace. Irrlicht engine podporuje velké množství běžných speciálních grafických efektů od mlhy, částicových efektů až po dynamické nasvětlování a stíny. Nabízí širokou škálu 3D formátu s podporou statických i animovaných modelů.

Irrlicht engine má detailní dokumentaci se spoustou příkladů a tutoriálů. Není omezen jen na C++, ale existují i portace pro jiné jazyky. Jedna z nich je Irrlicht.NET, která umožňuje použít rozhraní Irrlicht enginu v jakémkoliv .NET jazyce.

2.1.2 Crystal Space

Crystal Space[8] je plnohodnotné SDK (Software Development Kit), napsané v C++, a poskytující 3D grafiku v reálném čase, s větším zaměřením na hry samotné. Jedná se o multiplatformní rozhraní podporující Windows, Linux a Mac OSX, vydané pod licencí LGPL (Lesser General Public License). Projekt Crystal Space SDK je tvořen dvěma hlavními komponentami, samotným Crystal Space a CEL (Crystal Entity Layer). Crystal Space poskytuje konfigurovatelný renderovací engine založený na OpenGL, 3D zvuk, fyziku, uživatelský vstup a grafické uživatelské rozhraní (GUI). Chybí zde však správa entit, ta je zajištěna prostřednictvím CEL. Jedná se o množinu pluginů a aplikací postavených nad Crystal Space SDK, která poskytuje řadu běžně používaných abstrakcí pomáhajících při vývoji her.

Pro kolize a dynamiku je využíváno open source engine ODE (Open Dynamics Engine). Jsou zde implementovány standardní shadery jako normálové mapování, parallax (displacement) mapping a hardware skinning. Vlastní uživatelsky definované shadery lze implementovat v jazyku Cg nebo pomoci značkovacího jazyka XML. Pro práci s animovaným *mesh* je použita knihovna CAL3D (3D Character Animation Library). Jedná se o platformně nezávislou knihovnu napsanou v C++ a určenou pro skeletální animaci. Crystal Space má podporu pro renderingu 2D a 3D zvuku skrze rozhraní DirectSound, ALSA (Advanced Linux Sound Architecture), OSS (Open Sound System) a CoreAudio. Scénu je možné exportovat z modelovacích nástrojů Blender a 3D Studio Max.

K SDK Crystal Space je dostupný uživatelský manuál a dokumentace API, plně dokumentující rozhraní, třídy a funkce dostupné v SDK. Uživatelský manuál obsahuje popis jednotlivých komponent SDK, tutoriály a další technické články.

2.2 Komerční enginy

Komerční enginy ve srovnání s nekomerčními poskytují naopak komplexní řešení pro vývoj počítačových her. Kromě samotného vykreslovacího jádra, které většinou využívá nejnovějších technologií pro real-time vykreslování grafiky, obsahují zároveň vlastní fyzikální engine, popřípadě implementují jeden z komerčně dostupných enginů. Nechybí vlastní implementace umělé inteligence a také, u většiny enginů, podpora síťové hry a zvuku.

2.2.1 Source

Source[9] engine, jehož autorem je společnost Valve, je obecně uznáván jako jedno z nejvíce flexibilních, komplexních, a výkonných herních vývojových prostředí. Source engine je implementovaný v C++. Mezi podporované platformy patří PC a XBox 360.

Source engine, obr. 1, je přednostně určen pro rendering rozsáhlých areálů a venkovních detailních scén, doplněných propracovanou fyzikou simulující reálný svět. Nechybí ani sofistikovaný systém umělé inteligence, s možností emulace lidských smyslů - zraku a sluchu, umožňujících najít a identifikovat různé objekty. Pokročilá charakterová animace dovolující jazykově nezávislou artikulaci tváře a simulaci svalů. Source engine zahrnuje vlastní sadu digitálních audio technologií pro vytváření dynamických zvukových scén uvnitř herního světa. Vysoce výkonné renderovací jádro, založené na shaderech, dovoluje vytvářet rozsáhlé komplexní scény. Využívá pokročilých technologií dnešních multijádrových procesorů, jako je SIMD, a skrze DirectX také GPU. Součástí je také vlastní rozhraní pro síťovou komunikaci.

Obrázek 1: Source engine.

2.2.2 id Tech

V případě id Tech, obr. 2, se jedná spíše o sérii enginů, jejichž autorem je společnost id Software. Poslední verzí je id Tech 4[10] engine, a jeho nástupce id Tech 5, který je v současné době ve vývoji. Engine id Tech 4 je implementován v C++ a jeho renderovací jádro je založeno na OpenGL. Jedná se o velice svižný engine, využívající výhod moderních multijádrových procesorů a GPU. Po grafické stránce přinesl jako první ve své sérii per-pixel nasvětlování, normálové mapování a specular highlighting. Zvuková část je implementovaná za využití DirectX.

Výsadou těchto enginů je široká podpora různých platforem - PC, Mac OSX, Linux, Xbox, Xbox 360, PS3. Jsou doplněny o exportéry ze známých modelovacích a animačních nástrojů - 3D Studio Max, Maya a Lightwave.

2.2.3 CryENGINE®3

CryENGINE®3 [11], obr. 3, se řadí mezi nejpokrokovější enginy dnešní doby. Autorem tohoto enginu je společnost Crytek. Jedná se o kompletní herní vývojové řešení,

Obrázek 2: id Tech engine.

využívající moderních multi-core technologií, zahrnující pokročilou real-time 3D grafiku (DirectX 9, DirectX 10 a DirectX 11), fyziku, umělou inteligenci, zvuk a síťovou komunikaci. Mezi podporované platformy patří PC, PlayStation 3 a Xbox 360. Součástí je výkonný editor CryENGINE®3 Sandbox™ dovolující vytvářet herní prostředí v reálném čase, nebo-li WYSIWYP („What You See Is What You Play“).

Renderovací jádro se řadí mezi technologicky nejvyspělejší. Poskytuje možnost plynulých přechodů mezi indoor a outdoor scénami. Dostupná je celá řada speciálních grafických efektů a grafických technologií, pomoci kterých je dosaženo téměř foto-realisticky a přirozeně vypadajících scén. Využita je řada nasvětlovacích algoritmů od HDR (High Dynamic Range), umožňující rendering scény ve vyšších světelných kontrastech a tím dosažení větší realističnosti, až po takzvané „odložené nasvětlování“, neboli Deferred Lighting. Deferred Lighting je metoda dovolující per-pixel nasvětlování scény s velkým množstvím dynamických zdrojů světel. Flexibilní systém denního času dovoluje dynamicky měnit pozici slunce a měsíce ve scéně, a v závislosti na tom pak nasvětlování a atmosférické efekty. Propracovaná fyzika umožňuje vytvoření plně interaktivního a zničitelného prostředí, které dovoluje procedurální destrukci a deformaci velké části herního světa a objektů v něm. CryENGINE®3 poskytuje také vlastní zvukový systém plně integrovaný s herním prostředím. Ten umožňuje mimo jiné například audio odezvy na interaktivní fyziku herního světa a prostředí.

Obrázek 3: CryENGINE®3 .

3 Použité technologie

Jako primární aplikační programové rozhraní pro vykreslování grafiky bylo v této práci použito Microsoft Direct3D, které je součástí Microsoft DirectX. Je dostupná také alternativa OpenGL, upřednostněno však bylo DirectX z důvodu většího zaměření právě na počítačové hry.

3.1 DirectX

DirectX[3] je kolekce nízkoúrovňových aplikačních programových rozhraní (API), vytvořená společností Microsoft Corporation, za účelem manipulace s multimédií. Je speciálně určen pro programování počítačových her na platformě Microsoft Windows, Xbox a Xbox 360. Mezi hlavní rozhraní dostupné v DirectX:

- Direct3D (zobrazování hardwarově akcelerované 3D grafiky)
- Direct2D
- DirectInput
- DirectWrite
- DirectCompute
- DirectSound3D

Starší varianty těchto rozhraní DirectPlay, DirectSound, DirectMusic a DirectShow jsou v DirectX SDK stále dostupné, ale jsou již označeny jako zastaralé.

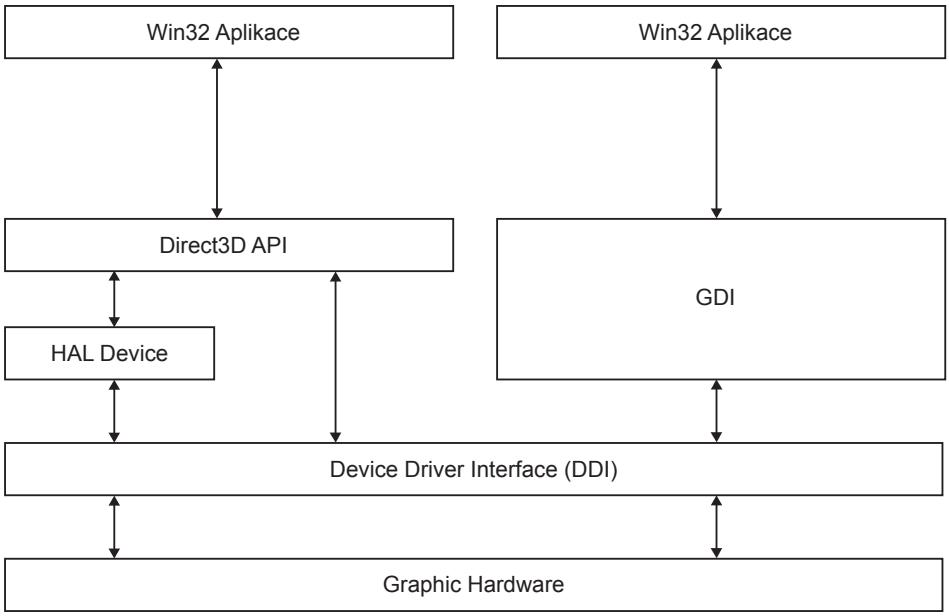
3.2 Direct3D

V aplikacích, kde je důležitá efektivita vykreslování, jako jsou počítačové hry, je použito funkcí, výše zmíněného rozhraní Direct3D, které využívá hardwarovou akceleraci, pokud je dostupná na grafickém adaptéru, celé 3D renderovací pipeline. Direct3D umožňuje využít pokročilé schopnosti grafické karty, zahrnující z-buffering, anti-aliasing, alpha blending, mipmapping a atmosférické efekty.

3.3 Architektura Direct3D

Hlavním úkolem API Direct3D je zprostředkovat komunikaci mezi aplikací a ovladačem grafického hardwaru. Direct3D rozhraní stojí nezávisle vedle GDI (Graphics Device Interface), viz obr. 4. Direct3D aplikace může existovat současně vedle GDI aplikace a obě mají přístup ke grafickému hardwaru přes ovladač grafické karty.

HAL (Hardware Abstraction Layer) představuje specifické rozhraní pro grafický hardware dodávané výrobcem. Direct3D toto rozhraní používá pro přímou komunikaci s grafickým adaptérem. HAL implementuje pouze funkce dostupné na adaptéru, pokud adaptér nějakou funkci neimplementuje, HAL tuto funkci nereflektuje jako schopnost hardwaru - neprovádí emulaci. Aplikace komunikuje s HAL skrze Direct3D.



Obrázek 4: Schéma začlenění DirectX do operačního systému.

4 Specifikace zadání

Tato práce je základem pro vytvoření herního enginu a následně počítačové hry, s pracovním názvem *Heroes*, která bude na něm postavená. Tento engine nemá být zaměřen na širší spektrum žánrů, jako komerční enginy, ale pouze na konkrétní typ počítačové hry. Zaměřením se engine bude soustředit na hry strategického žánru a případně RPG. Engine bude implementován jako samostatná jednotka oddělená od aplikace (hry), ta bude jako oddělená aplikace tento engine následně využívat.

4.1 Vizuální vzhled

Mým cílem v této práci je zaměřit se na vizuální stránku enginu. Tedy na vytvoření grafické části enginu. Po vizuální stránce bude hra tvořena převážně jednoduchým terénem. Na povrchu se mohou vyskytovat statické renderovatelné objekty. Engine bude zvládat jednoduché techniky optimalizace scény.

Do budoucna je v plánu tento engine výrazně rozšířit a více optimalizovat. Budou přidány částicové efekty a post-process efekty. Počítá se také s dynamickým nasvětlováním scény a realizací stínování.

4.2 Herní koncept

Počítačová hra *Heroes*, na jejímž základu budeme v bakalářské práci pracovat, bude vycházet z kombinací klasického konceptu hry typu RPG, kdy hráč ovládá jediného hrdinu a strategické hry, ve které je záměr poražení nepřátelské strany zničením jeho hlavní budovy nebo důležité jednotky. Předpokládáme, že hráč bude podobně jako ve hře Diablo firmy Bilzzard, ovládat svého hrdinu myší a určovat jeho cíle cesty kliknutím do mapy. Postava na toto místo dojde nebo zaútočí na nepřátelskou postavu, která se na tomto místě nachází. Za zničené nepřátelské jednotky bude postava dostávat peníze, za které si bude dokupovat vylepšení. Herní mapa bude po technologické stránce podobná mapě strategické hry. Bude obsahovat množství překážek a její území bude rozděleno mezi jednotlivé strany na kterých bude mít každá strana klíčovou budovu nebo jednotku, jejíž porážkou dojde k výhře jedné ze stran. Hra bude oživena tím, že jednou za určitý časový úsek vyjde z hlavních budov všech hráčů určité množství, na hráčích nezávislých, jednotek vyslaných na cizí základny. Cílem hrdinů (hráčů) bude pomoc těmto jednotkám v přesunu na nepřátelskou stranu. Zpravidla vítězí pak ti hráči, kterým se podaří probojovat až do nepřátelské základny, aniž by přišli o svou vlastní. Hra bude umožňovat, aby zároveň ve hře mohlo hrát více hráčů, kteří by měli každý svého hrdinu. Postavy vybíhající ze základen každé strany budou postupně silnější, aby se kompenzoval nárůst síly hrdinů. Tyto postavy se budou řídit jednoduchým plánem cest a předem budou mít nastavené své chování.

4.3 Technicka specifikace

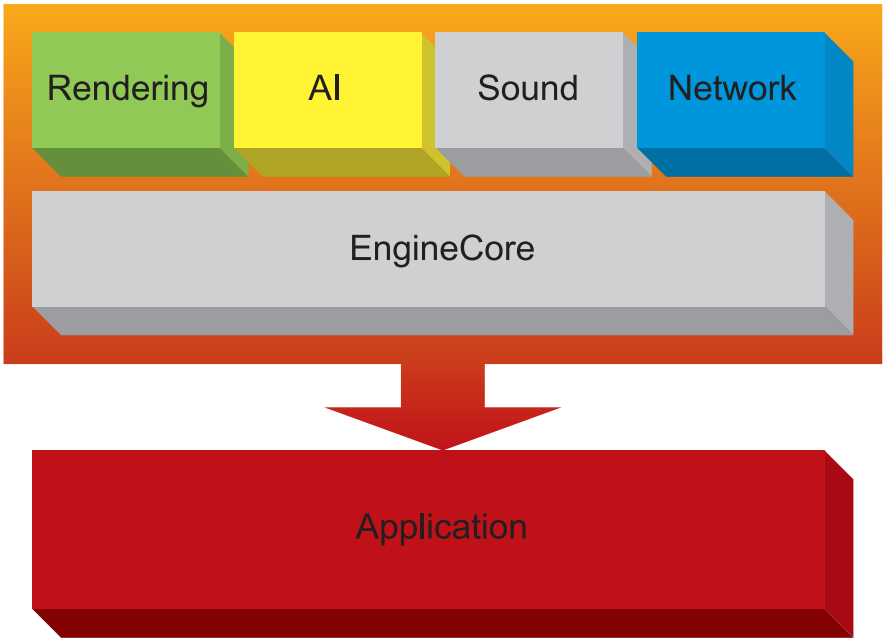
Pro charaktery a interaktivní objekty bude podporován speciálně upravený 3D formát, doplněný o specifikaci vlastností dané entity. Pro tento formát bude vytvořen v budoucím vývoji editor, umožňující doplnění o tyto specifické parametry. Tento formát bude založen na *DirectX X File Format*. Společně s geometrií budou uchována také skinned mesh data v případě animovaných modelů a případně i kolizní geometrie.

Pro reprezentaci mapy světa bude v budoucnu vytvořen speciální formát kompatibilní jen s tímto enginem. Tento formát bude uchovávat geometrii terénu, odkazy a pozice na entity vyskytující se ve scéně, mapu cest AI a případně script AI. K vytváření a editaci bude sloužit editor vytvořený speciálně pro tento formát.

5 Analýza a návrh implementace enginu

Jako základ architektury bylo zvoleno modulové schéma, tedy každá dílčí část enginu je nebo bude tvořena samostatným modulem, které mezi sebou komunikují. Těmito moduly se myslí grafický engine, umělá inteligence, síť a zvuk. Návrh této architektury je znázorněn na obrázku 5. Implementace uvedených modulů se předpokládá pomoci dynamicky linkovaných knihoven (DLL). V budoucím vývoji je také v plánu přidání dalšího modulu, na obrázku znázorněného jako Engine Core, který bude propojovat všechny ostatní moduly do jednoho celku. Důvodem pro vytvoření tohoto modulu je obalení komunikace mezi jednotlivými moduly enginu a snadnější použití v klientské aplikaci. Klientskou aplikací, znázorněnou na obr. 5, je myšlena samotná hra.

V této práci se zaměřuji pouze na základ grafického modulu. Ten má na starosti vizuální stránku enginu. V současné verzi se jedná o zobrazení jednoduchého herního světa. V budoucím vývoji se předpokládá přidání pokročilejších nasvětlovacích metod, vylepšení metod optimalizace vykreslování a zprávy scény, přidání grafických a částicových efektů, a implementace skeletální animace.

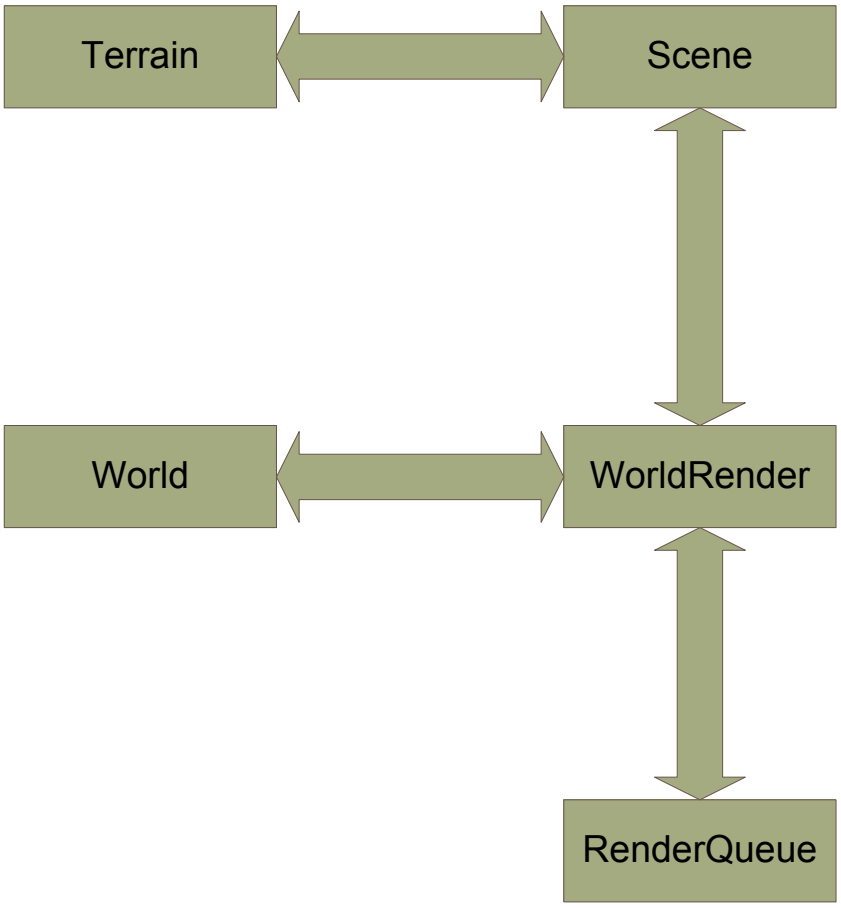


Obrázek 5: Schéma architektury enginu.

Modul AI (Artificial intelligence) je implementací metod umělé inteligence. Modul Audio bude implementovat zvukovou stránku enginu. V budoucí implementaci tohoto modulu se předpokládá využití audio knihovny FMOD společnosti Firelight Technologies, nebo multi-platformní knihovny OpenAL. Modul Network bude implementovat vytváření serveru a síťovou komunikaci mezi klienty.

5.1 Grafický modul

Grafický modul implementuje samotnou vizuální stránku enginu. Stará se o vizualizaci všech objektů nacházejících se ve scéně a také o zprávu zdrojů, tedy načítání a uchovávání objektů v paměti. Určuje, jakým způsobem budou zdroje zpravovány, jestli budou načteny do video paměti grafické karty, nebo budou umístěny v systémové paměti.



Obrázek 6: Základní schéma grafického modulu.

Na obr. 6 je znázorněno základní schéma grafického modulu. Vyobrazeny jsou jen základní objekty starající se o zobrazování grafiky a správu zdrojů. Objekt *World* je základním objektem grafického modulu. Stará se o instanciování a inicializaci ostatních objektů na obr. 6. Implementuje funkci pro inicializaci celého grafického modulu. Skrze rozhraní klientská aplikace komunikuje právě s tímto objektem. K rozhraní ostatních objektů už klientská aplikace přístup nemá. Volání funkcí rozhraní ostatních objektů probíhá interně v modulu.

Tento modul je implementován z velké části za využití funkcionality aplikačního programového rozhraní Direct3D 9.0c. Skrze rozhraní *zařízení Direct3D* je realizováno jak zobrazování grafiky tak i načítaní a správa zdrojů v paměti. Alokace paměti pro určité typy zdrojů je realizována také skrze rozhraní zařízení Direct3D. Témě všechny objekty v tomto modulu toto zařízení využívají. Za správné vytvoření, inicializaci a uvolnění rozhraní zařízení Direct3D je zodpovědná klientská aplikace. Ta předává instanci zařízení grafickému modulu při inicializaci.

5.2 Objekt Terrain

Jedním z objektů vytvořených při první inicializaci modulu je *Terrain*. Na základě vstupních parametrů vygeneruje geometrii terénu. Má na starost také alokaci a správu paměti pro tuto geometrii. Jako vstup pro generování jsou rozměry terénu a hustota geometrie. Aby bylo možné realizovat vykreslování rozsáhlejšího terénu, je geometrie rozdělena na sektory. Toto rozdělení na sektory umožňuje, za použití optimalizačních algoritmů, vykreslovat jen tu část geometrie, která je skutečně viditelná. Rozdělení geometrie na sektory probíhá při samotném generování. Geometrie každého sektoru je pak v objektu Terrain uchována v samostatném bufferu. Společně s generováním geometrie jednotlivých sektorů je načtena také výška jednotlivých vrcholů z výškové mapy. Výšková mapa je textura ve stupních šedi, kde intenzita jasu *texelu* definuje výšku geometrie. Pomoci výškové mapy jsou tak modelovány nerovnosti terénu. Při ukončení aplikace se Terrain stará o uvolnění paměti alokované pro sektory terénu a ostatní zdroje.

5.3 Objekt Scene

Zobrazovaný svět je z důvodu optimalizace rozdělen na menší celky, které jsou jednotlivě podle potřeby vykreslovány. Toto rozdělení světa je reprezentováno právě tímto objektem. Objekt *Scene* je jedním z nejdůležitějších objektů, jedná se o stromovou strukturu reprezentující celou scénu. Vnitřně je Scene implementován pomoci QuadTree. QuadTree je stromová struktura rekurzivně dělící prostor do čtyř kvadrantů. Jakýkoliv objekt (entita), který se má nacházet ve scéně, a nemusí se jednat pouze o renderované objekty, musí být vložen do tohoto stromu. Pozice na kterou je objekt do stromu vložen přímo závisí na pozici objektu ve scéně. Ve fázi vykreslování jsou objekty vybírány z této struktury podle toho, jakou část scény je potřeba vykreslit.

Do stromu jsou vloženy také jednotlivé sektory terénu, podle toho ve kterém místě scény mají být vykresleny. Fyzicky jsou ale jednotlivé instance terénu uloženy v objektu Terrain, do stromu jsou vloženy pouze reference.

5.4 Objekt WorldRender a RenderQueue

Samotná funkcionalita pro vykreslování scény je implementována v objektu WorldRender. Vykreslování je prováděno v nekonečné smyčce. Ta je realizovaná klientskou aplikací a funkce pro vykresleni snímku, nacházející se ve WorldRender, je v ní v každém

průchodu cyklu volána. Na nejnižší úrovni je vykreslování grafiky ve WorldRender implementováno pomoci shaderů grafické karty realizovaných za použití HLSL (High Level Shader Language). Využit je Shader Model 3.0.

Objekt WorldRender úzce spolupracuje s objektem RenderQueue. Hlavním úkolem RenderQueue je shromáždit renderovatelné entity před samotným procesem vykreslování. Objekt RenderQueue je v základu implementován jako fronta. Před začátkem vykreslování je za použití *algoritmu určení viditelnosti* rozhodnuto, které entity ve scéně jsou viditelné pozorovatelem a měli by být vykresleny. Tyto vybrané entity jsou následně vkládány do RenderQueue. Důležitou vlastností RenderQueue je, že vkládané entity jsou uvnitř udržovány v setřízené formě podle typu entity. Poté co je objekt RenderQueue zaplněn, přejde se k samotnému vykreslování. Entity jsou z RenderQueue postupně vybírány a vykreslovány. Protože jsou entity v RenderQueue udržovány v setřízené formě, je vždy vykreslena skupina entit stejného typu najednou, což eliminuje četnost nastavování stavů vykreslování a množství přenášených dat do paměti grafické karty. Na začátku následujícího průchodu vykreslovací smyčkou jsou entity z RenderQueue hromadně vyjmuty.

5.5 Schéma komunikace

Komunikačním schématem je v tomto případě myšlena komunikace mezi klientskou aplikací a grafickou částí enginu. Základní schéma této komunikace je zobrazeno na obr. 7. Poté co je na začátku aplikace vytvořena instance grafické části je provedena inicializace rozhraní voláním funkce *Initialize*. Součástí inicializace je také načtení mapy světa z externích zdrojů. Načtením mapy světa je vytvořena geometrie terénu (Terrain) a na základě toho je inicializován strom scény (Scene). V této fázi jsou také do paměti nahrány všechny potřebné textury a případně ostatní zdroje potřebné při renderingu scény. Tato inicializace je provedena jen jednou po spuštění aplikace.

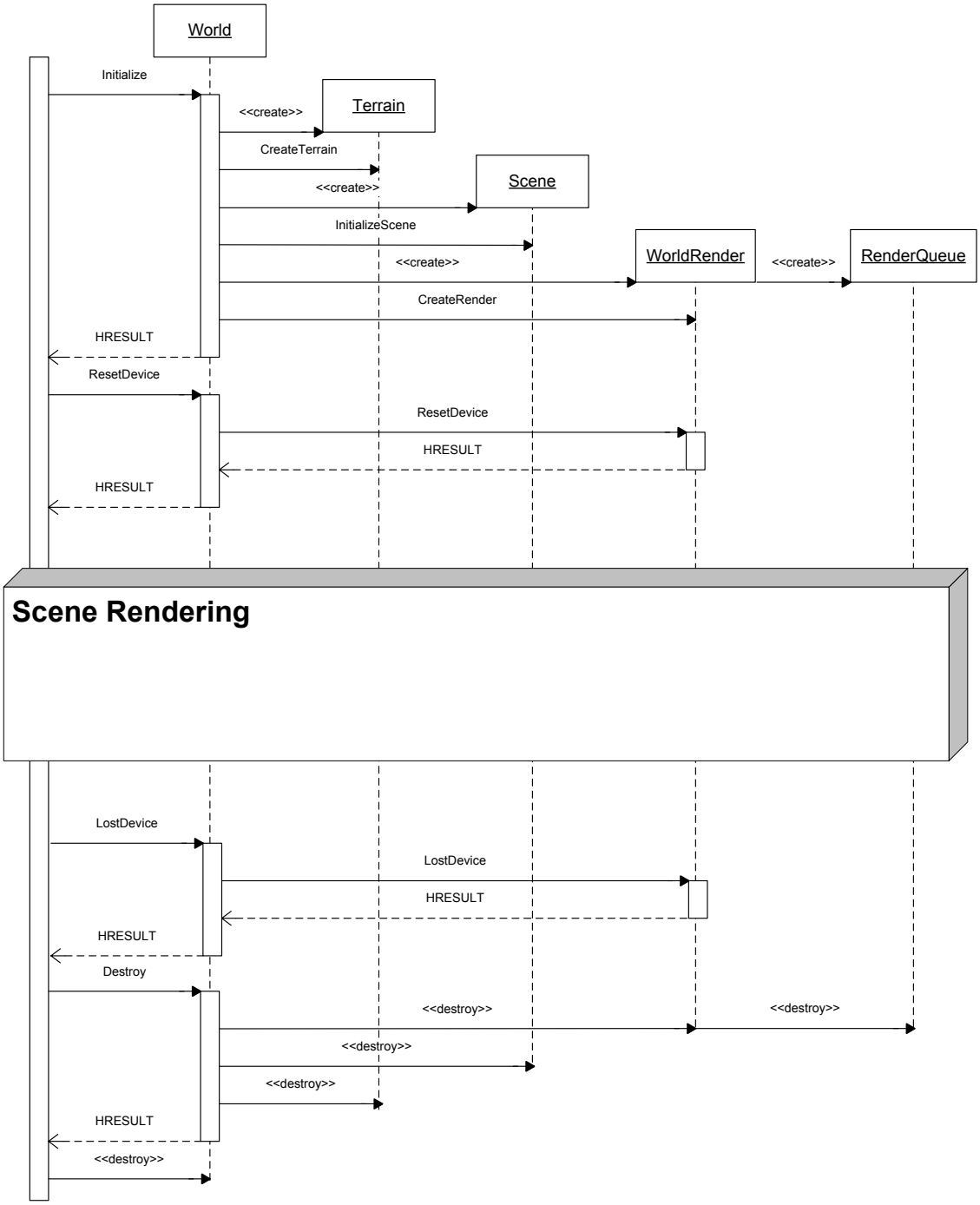
Bezprostředně po inicializaci následuje volání *ResetDevice*. V tomto momentě jsou vytvořeny a inicializovány všechny zdroje fyzicky umístěné v paměti grafické karty. Jedná se o speciální skupinu zdrojů, které musí být znovu vytvořeny a inicializovány po případné ztrátě *zařízení Direct3D*. Tato metoda je vykonána pokaždé, co během vykonávání renderovací smyčky dojde ke ztrátě *zařízení Direct3D*, aby byly zdroje znovu obnoveny.

V případě ztráty *zařízení Direct3D*, ještě před voláním samotné funkce ResetDevice, je třeba uvolnit zdroje, které mají být znovu obnoveny. Tento úkol obstarává funkce *LostDevice*. Uvolňuje všechny zdroje vytvořené v ResetDevice z paměti.

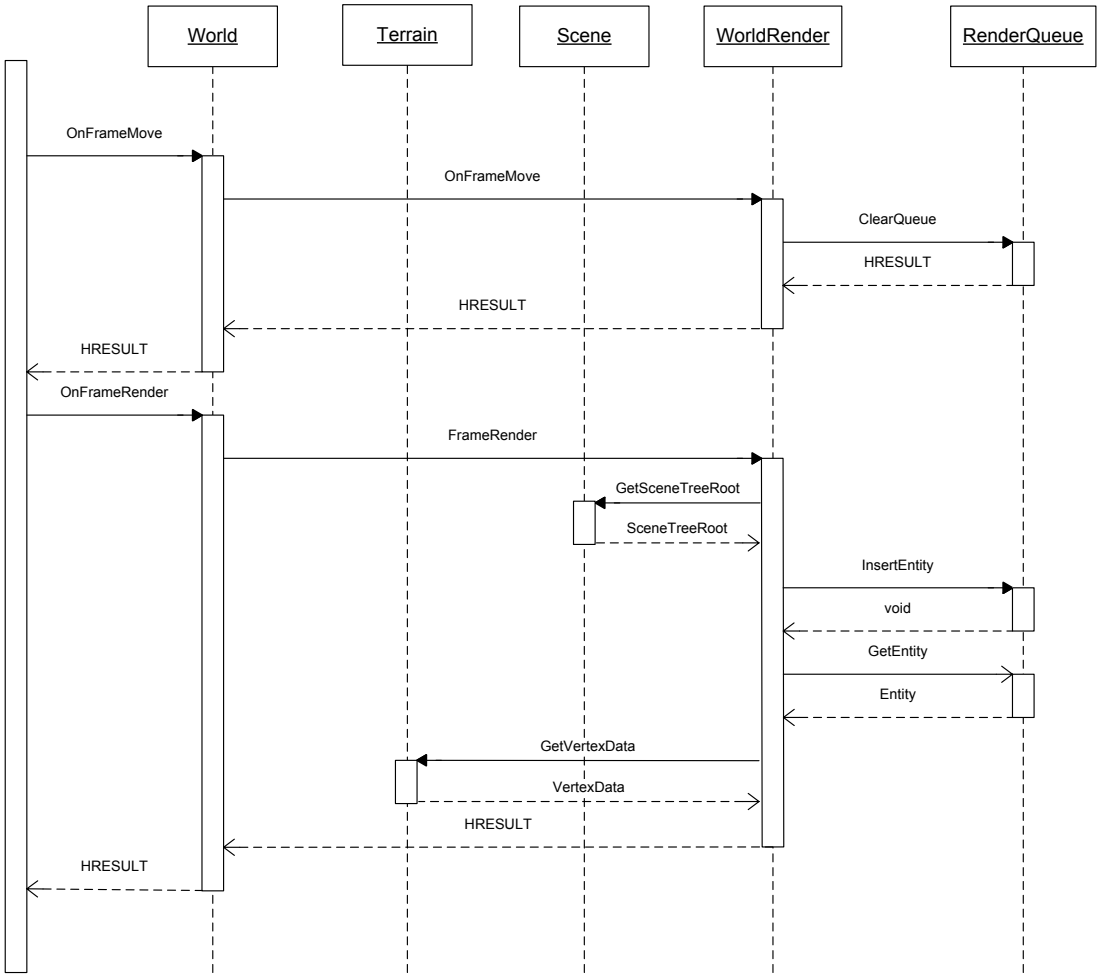
Na konci aplikace jsou zrušeny všechny vytvořené zdroje. Jako první je volána metoda LostDevice, pro uvolnění zdrojů vytvořených v ResetDevice. Následně jsou uvolněny všechny zbývající zdroje a interní instance objektů voláním *Destroy*.

Scene Rendering představuje vykonávání samotné renderovací smyčky. Tato komunikace je podrobněji znázorněna na obr. 8. O samotné vykonávání smyčky se stará klientská aplikace, ze které jsou pak volány funkce pro rendering snímků. Samotný rendering probíhá voláním *FrameRender*. V této fázi jsou ze stromu scény vybírány entity, které

mají být vykresleny, a vkládány do RenderQueue. Na konci této metody jsou všechny entity v RenderQueue postupně vykreslovány.



Obrázek 7: Základní komunikace mezi klientskou aplikací a grafickou částí enginu.



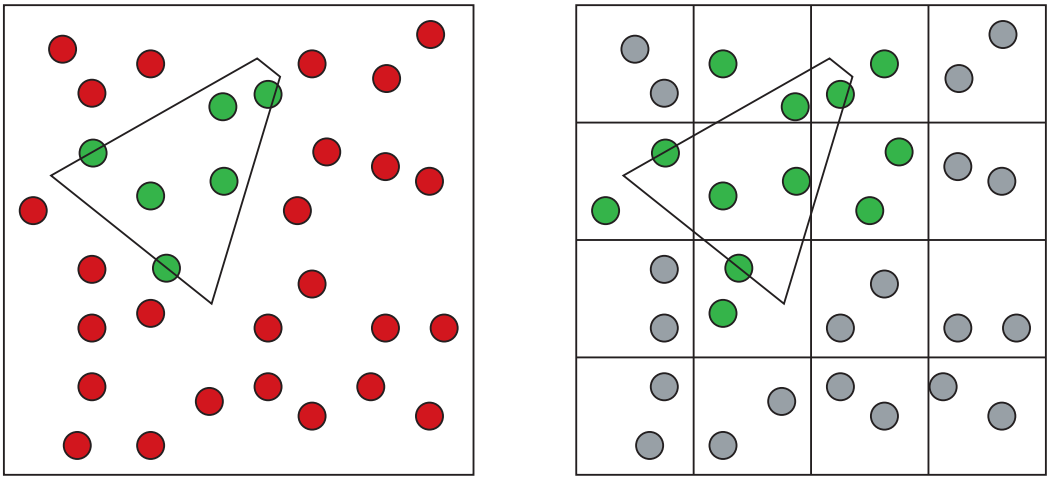
Obrázek 8: Základní komunikace mezi klientskou aplikací a grafickou částí enginu ve fázi vykreslování.

6 Návrh implementace reprezentace scény

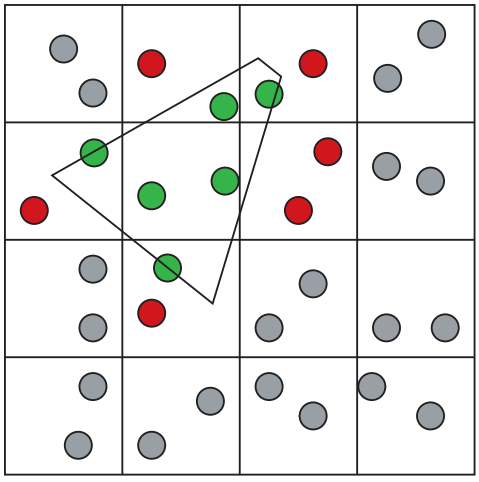
Součástí rozsáhlých scén je velký počet objektů, které je třeba vykreslovat. Ne všechny jsou ovšem v jednom momentě viditelné pozorovatelem. Metoda *Frustum Clipping*, popsaná v kapitole 7.2.1, je efektivní způsob jak vyloučit objekt z vykreslování, pokud zrovna není viditelný. Pokud se ovšem ve scéně nachází stovky objektů, u kterých je třeba rozhodnout, zda-li jsou viditelné, není zrovna efektivní tuto metodu aplikovat postupně na všechny. Na obr. 9 vlevo je takováto scéna pro ilustraci zobrazena. Pro každý objekt je třeba provést zvlášť test viditelnosti vzhledem k pohledovému kuželu, vyloučené objekty jsou pak vyznačeny červeně. Objekty zobrazeny zeleně naopak testem viditelnosti prošli a budou vykresleny. Pohledový kužel je tvořen šesti rovinami, a pro každý objekt je v podstatě třeba vykonat sadu testů vzhledem k těmto šesti rovinám. Spíše než k optimalizaci by toto vedlo ke snížení výkonu aplikace. Scénu a objekty v ní je proto potřeba nějakým způsobem reprezentovat tak, aby se zjednodušil výběr neviditelných objektů a nebylo potřeba najednou testovat všechny objekty scény. Tohoto je docíleno rozdělením prostoru celé scény na sektory. Každému sektoru pak náleží určitá podmnožina objektů. Toto rozdělení scény na sektory je demonstrováno na obr. 9 vpravo. Test viditelnosti je pak proveden pro celé sektory. Pokud je sektor zcela nebo z části viditelný, tedy nachází se uvnitř pohledového objemu, jsou všechny objekty náležící do tohoto sektoru vykresleny. Objekty ležící v sektorech nacházejících se mimo pohledový kužel jsou z vykreslování vyloučeny bez dalších testů. Na obr. 9 jsou objekty které byly z testováni zcela vyloučeny zobrazeny šedě.

Pro větší preciznost je do budoucna v plánu tuto metodu vylučování dále rozšířit následujícím způsobem. Pokud se testovaný sektor nachází v pohledovém kuželu jen z části, může být test viditelnosti proveden dále na každém objektu, náležejícímu tomuto sektoru, zvlášť. Toto rozšíření je demonstrováno na obr. 10. Tímto jsou do procesu vykreslování zahrnuty jen ty objekty, které jsou skutečně viditelné.

Funkcionalita reprezentace scény je implementována v objektu *Scene*, viz kapitola 5. Samotný algoritmus určování viditelnosti je pak implementován v objektu *WorldRender*. Ten rekurzivně prochází strukturu *Scene* a vybírá viditelné objekty.



Obrázek 9: Nalevo je scéna bez rozdělení na sektory a pro každý objekt je zvlášť proveden test viditelnosti. Napravo jsou naopak objekty přiřazeny jednotlivým sektorům scény. Test viditelnosti je pak proveden jen pro tyto sektory.



Obrázek 10: Pro objekty v sektorech, nacházející se z části v pohledovém kuželu, je test viditelnosti dále proveden zvlášť. tímto je dosaženo větší preciznosti při vylučováni neviditelných objektů.

7 Implementace metody Frustum Clipping

Rozsáhlá scéna je ve většině případech tvořena velkým množstvím objektů. Vykreslování takových to rozsáhlejších scén klade nemalé nároky na hardware. Frekvence s jakou je grafická karta schopna zobrazovat jednotlivé snímky je vyjádřena v FPS (Frames Per Second). Vykreslení všech objektů ve scéně najednou by vedlo k rapidnímu snížení FPS. S toho důvodu existují metody pro optimalizaci vykreslování, které jsou založeny na principu vylučování (nebo-li takzvaný *culling*) objektů, které nejsou v danou chvíli viditelné pozorovatelem. Takto vyloučené objekty pak není nutné vykreslovat. V této práci byla pro vylučování neviditelných objektů použita metoda Frustum Clipping. Metoda vyloučí objekt z vykreslování na základě toho, zda je ve scéně viditelný.

7.1 Vylučování na úrovni GPU

Grafická karta sama některé z metod pro určování viditelnosti implementuje. V takovém případě mluvíme o *určování viditelnosti na úrovni GPU*, viz [2]. Jedná se o metodu *back-face culling*, nebo-li metodu vylučování odvrácených ploch. Metoda *back-face culling* na základě jednoduchého testu určuje zda je plocha polygon viditelná z pozice pozorovatele, a podle toho rozhodne, zda bude polygon rasterizován. Další metoda je z-buffering. U této metody je test viditelnosti prováděn na nejnižší možné úrovni - provádí se na úrovni samotných pixelů. Při renderingu objektů si grafická karta uchovává hodnotu hloubky každého pixelu, na základě které je pak pixel vyloučen z dalšího zpracování. Implementována je také metoda Frustum Clipping. Tímto testem hardware kompletně vyloučí polygon, pokud se nachází mimo pohledový jehlan, viz obr. 15.

Metody implementované na úrovni GPU mají tu nevýhodu, že pracují na velmi nízké úrovni. To znamená, že vylučování neviditelných objektů provádějí na úrovni polygonů, nebo samotných pixelů. Proto pro vyloučení komplexnějšího objektu skládajícího se z většího množství polygonů musí být na GPU provedená spousta testů, a to i přesto, že objekt se ve scéně nachází kompletně mimo pohledový jehlan. Vezmeme-li v úvahu například Back-Face Culling, tak ten je proveden na vyšší úrovni než Z-Buffering, který vyloučení neviditelného polygonu provádí na základě každého pixelu. Je zřejmé, že nechat takové vylučováni neviditelných objektů, v případě rozsáhlé scény na GPU, je značně neefektivní. Řešením je provést tyto testy ještě na vyšší úrovni, tedy na úrovni CPU.

7.2 Vylučování na úrovni CPU

Ještě před tím, než je samotný objekt vykreslen, je proveden test viditelnosti, tedy test při kterém je rozhodnuto, jestli se objekt nachází uvnitř, nebo mimo pohledový jehlan. Pro urychlení tohoto testu jsou objekty obaleny jednoduššími geometrickými tvary, které aproximují tvar samotného objektu. Na základě těchto aproximačních objektů se pak testuje, zda je objekt viditelný. Geometrické tvary pro aproximaci tvaru objektů ve scéně se pak volí takové, aby proces výpočtu byl co nejefektivnější, a zároveň aby co nejpřesněji kopírovali tvar původních objektů. Nejčastěji používané jsou koule (*Bounding Sphere*) viz

obr. 11, objektově orientovaný kvádr (*OOBB - Object Oriented Bounding Box*) viz obr. 12, a osově orientovaný kvádr (*AABB - Axis Aligned Bounding Box*) viz obr. 13.

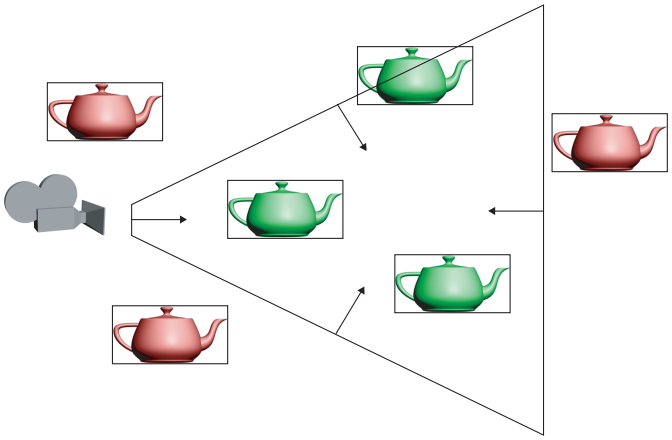
Obrázek 11: Bounding Sphere

Obrázek 12: OOBB - Object Oriented Bounding Box

Obrázek 13: AABB - Axis Aligned Bounding Box

7.2.1 Frustum Clipping

Implementace metody *Frustum Clipping* tvoří v podstatě základ metod optimalizace vykreslování v této práci. Metoda je použita hlavně jako způsob určování viditelných částí scény, tedy větších skupin entit, viz kapitola. 8.



Obrázek 14: Určování viditelnosti objektů ve scéně na základě pohledového jehlanu.

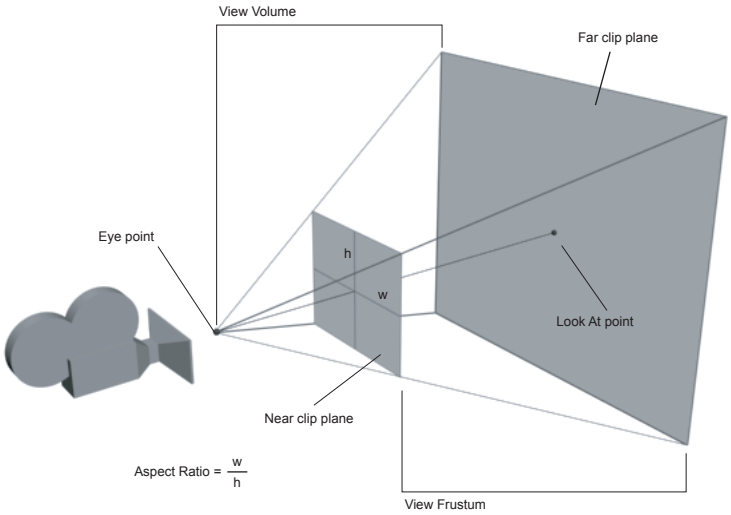
Obkjekty ve scéně, které jsou viditelné pozorovatelem, se nacházejí uvnitř pohledového jehlanu. Pohledový jehlan je tvořen šesti rovinami ve 3D prostoru, viz obr. 15, jejichž normálové vektory směřují dovnitř pohledového kužele. Rovina ρ je obecně definovaná rovnicí

$$\rho : n \bullet x + D = 0, x \in \mathbb{R}^3, \tag{1}$$

kde $n \in \mathbb{R}^3$ představuje *normálový vektor* této roviny a $D \in \mathbb{R}$ je vzdálenost roviny od počátku po směru normálového vektoru. Takto definovaná rovina na základě svého normálového vektoru rozděluje prostor na kladný a záporný poloprostor. O rovině ρ pak mluvíme jako o *hraniční rovině*. Směr normálového vektoru definuje *kladný poloprostor*. Celý test pak probíhá tak, že objekty ležící zcela, nebo z části uvnitř pohledového jehlanu, definovaného těmito šesti rovinami, jsou označeny jako viditelné a následně vykresleny. Objekty které leží mimo pohledový jehlan jsou z dalšího zpracování vyloučeny. Na obr. 14 jsou červeně znázorněny objekty vyloučené ze zpracování a zeleně vykreslované objekty.

Aby se objekt nacházel uvnitř pohledového jehlanu musí být pro všech šest rovin splněna podmínka, že testovaný objekt leží v kladném poloprostoru tvořeném danou rovinou. Tato podmínka je v podstatě založena na testování kolize objektu a roviny. Pro účely testování těchto kolizí jsou objekty aproximovány pomoci výše zmíněných bounding boxů, takže se pak testuje už jen kolize roviny a bounding boxu. Jsou různé metody pro výpočet této kolize.V této práci jsem použil metodu založenou na využití takzvaného *efektivního rádiusu* [2].

7.2.1.1 Efektivní rádius Pro test kolize koule a roviny stačí znát pouze vzdálenost středu koule od roviny a poloměr samotné koule. Test je pak založen jen na porovnání délky rádiusu této koule a vzdálenosti jejího středu od roviny. Pokud je vzdálenost středu koule menší než její rádius, došlo ke kolizi s rovinou.



Obrázek 15: Pohledový objem a pohledový kužel.

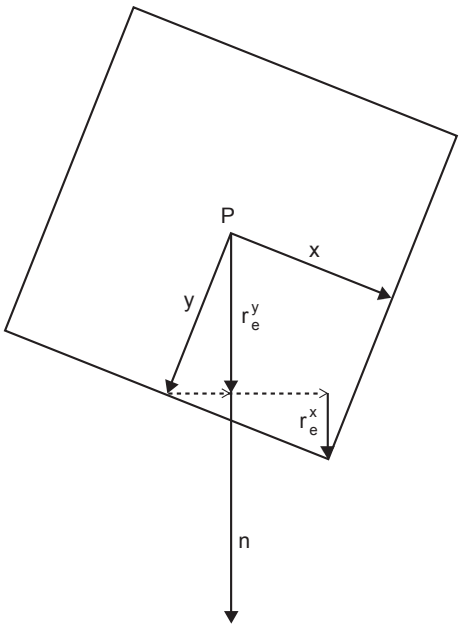
Test zda bounding box koliduje s rovinou je založen na podobném principu jako při výpočtu kolize koule s rovinou. Nepočítá se však s rádiusem koule ale s takzvaným *efektivním rádiusem* boxu. Efektivní rádius boxu je vzdálenost středu boxu od jedné ze stěn tohoto boxu. Poté co je znám efektivní rádius bounding boxu, je možné využít stejnou metodu jako při výpočtu kolize koule a roviny. Aby tato metoda ovšem fungovala podle předpokladu, je nutné aby vektor *efektivního rádiusu* byl kolineární s normálovým vektorem roviny. Podle obr. 16 je pak *efektivní rádius* r_e vypočítat jako

$$r_e = r_e^x + r_e^y = x \bullet n + y \bullet n \,, \tag{2}$$

kde n je *normalizovaný normálový vektor* roviny kolineární s vektorem *efektivního rádiusu* r_e. Na obr. 16 je výpočet *efektivního rádiusu* pro ilustraci zobrazen ve 2D, stejného principu se však využívá ve 3D, stačí jen přidat do výpočtu třetí rozměr. Výpočet využívá jedné z vlastností skalárního součinu zvané *skalární projekce*, na základě které projektujeme vektor x a y do vektoru n. Takto dostaneme vektory r_e^x a r_e^y, viz obr. 16.

V závislosti na tom, na jaké straně roviny se bounding box nachází, což je určeno směrem normálového vektoru roviny, závisí také znaménko *efektivního rádiusu*. Ten může být buď kladný nebo záporný. Pomoci znamének efektivního rádiusu je tedy možné určit, zda se bounding box nachází v kladné části poloprostoru nebo záporné.

7.2.1.2 Clipping Poté co je znám efektivní rádius, je výpočet, zda se testovaný box nachází uvnitř pohledového jehlanu, velice podobný tomu, jako bychom místo boxu uvažovali kouli. Jediný rozdíl je v tom, že efektivní rádius boxu musí být vypočítán pro každou ze šesti rovin tvořících pohledový jehlan. Poté co je znám efektivní rádius je třeba ještě vypočítat vzdálenost středu boxu od dané roviny. V podstatě se však jedná o určení nejkratší vzdálenosti bodu v prostoru od roviny.



Obrázek 16: Výpočet efektivního rádiusu za využití skalární projekce.

Uvažujme rovinu ρ definovanou její obecnou rovnicí $\rho : n \bullet x + D = 0, x \in \mathbb{R}^3$. Střed boxu označíme $P \in \mathbb{R}^3$. Pokud je daná rovina normalizovaná, tak se D stane *eukleidovskou vzdáleností* mezi počátkem prostoru a rovinou. Pokud pak rovinu ρ transformujeme tak, že počátkem se stane bod P, pak D bude rovno nejkratší vzdáleností mezi plochou a bodem P.

Normalizaci roviny zadané rovnicí $n \bullet x + D = 0, x \in \mathbb{R}^3$ lze vyjádřit jako

$$\frac{n_x x + n_y y + n_z z}{\|n\|_2} + D = 0$$

Dalším krokem je transformace (posunutí) prostoru tak, aby se bod P stal počátkem, přičtením bodu P

$$n\,(x + P) + D = 0$$

$$n \bullet x + D + n \bullet P = 0$$

položíme $n \bullet P = d$, dostaneme:

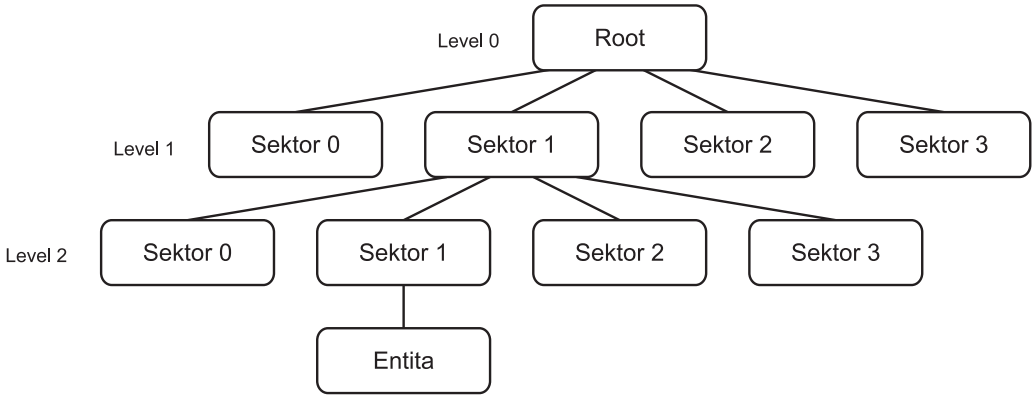
$$n \bullet x + D + d = 0$$

$$d = D + n \bullet P$$

kde d je hledaná minimální vzdálenost bodu P od roviny ρ. Porovnáním minimální vzdálenosti středu boxu d a efektivního rádiusu r_e pro každou ze šesti rovin pohledového kužele, lze pak zjistit, zda se box nachází uvnitř, nebo mimo pohledový kužel.

8 Implementace reprezentace scény

Způsob reprezentace scény je implementována v objektu Scene, viz kapitola 5. Pro efektivnější procházení jsou sektory scény udržovány ve formě stromové struktury *QuadTree*. QuadTree představuje v základu 2D organizaci prostoru. Pro povahu scény prezentované v této práci, která je tvořena jen terénem a objekty na něm, je tato stromová struktura dostačující. Jednotlivé uzly QuadTree jsou reprezentovány sektory, dělícími prostor herního světa na menší části. Každý sektor je rozdělen na další čtyři menší sektory tvořící potomky tohoto sektoru (uzlu). Svou rozlohou potomci kompletně vyplňují prostor svého rodiče. Příklad struktury QuadTree je demonstrován na obr. 17. V jednotlivých uzlech QuadTree jsou pak umístěny entity, nacházející se ve scéně.



Obrázek 17: Příklad struktury QuadTree.

Organizace sektorů s využitím QuadTree dovoluje rychlejší procházení a vybírání sektorů, které jsou viditelné pozorovatelem. Entity náležející do takto vybraných sektorů pak můžou být vykresleny, nebo jinak zpracovány.

Pro efektivní přístup k uzlům stromu je QuadTree implementován metodou přímého přístupu - Direct Access QuadTree Lookup. Tuto metodu definoval Matt Pritchard ve svém článku v [5]. Jedná se o hlavní optimalizační metodu přístupu do QuadTree vylepšující výkon hned ve třech věcech oproti tradiční implementaci: 1) eliminací nadbytečného procházení uzlů, 2) celkové snížení počtu instrukcí a cyklů CPU, 3) zrychlení procházení s minimálním přístupem do paměti, bez ohledu na to v jaké hloubce stromu se cílový uzel nachází.

Omezení tohoto přístupu je, že jednotlivé uzly v určitém levelu stromu musejí mít shodnou velikost. Před vytvořením QuadTree musí být také dopředu znám maximální level stromu. Vhledem k povaze herní scény v této práci jsou však tyto omezení přijatelná.

8.1 Implementace uzlu QuadTree

Každý uzel QuadTree má, jak už bylo zmíněno, čtyři potomky dělící prostor tohoto uzlu na čtyři shodně velké části, na které si uchovává odkazy. Každý uzel si uchovává také odkaz na svého rodiče. Jednotlivé entity, nacházející se ve scéně, jsou umístěny v uzlech QuadTree. Pro tento účel každý uzel obsahuje seznam entit, které mu náleží.

Prostorově je každý uzel QuadTree reprezentován osově srovnaným hraničním boxem (Axis Aligned Bounding Box), viz obr. 13 kap. 7.2. Ten představuje skutečný objem, jaký daný uzel zabírá fyzicky ve scéně. Tento objem je vypočítán na základě součtu objemů entit nacházejících se v tomto uzlu a objemů jeho potomků. Na základě objemu uzlu je detekována viditelnost tohoto uzlu a entit v něm ve scéně.

8.2 Vkládání entit do QuadTree

V případě vkládání libovolné entity do stromu se jedná o velmi rychlou operaci. Nejprve je určen level, do kterého bude entita umístěna a následné jsou vypočítány přesné souřadnice uzlu v daném levelu. Entita je pak přímo vložena do konkrétního uzlu stromu.

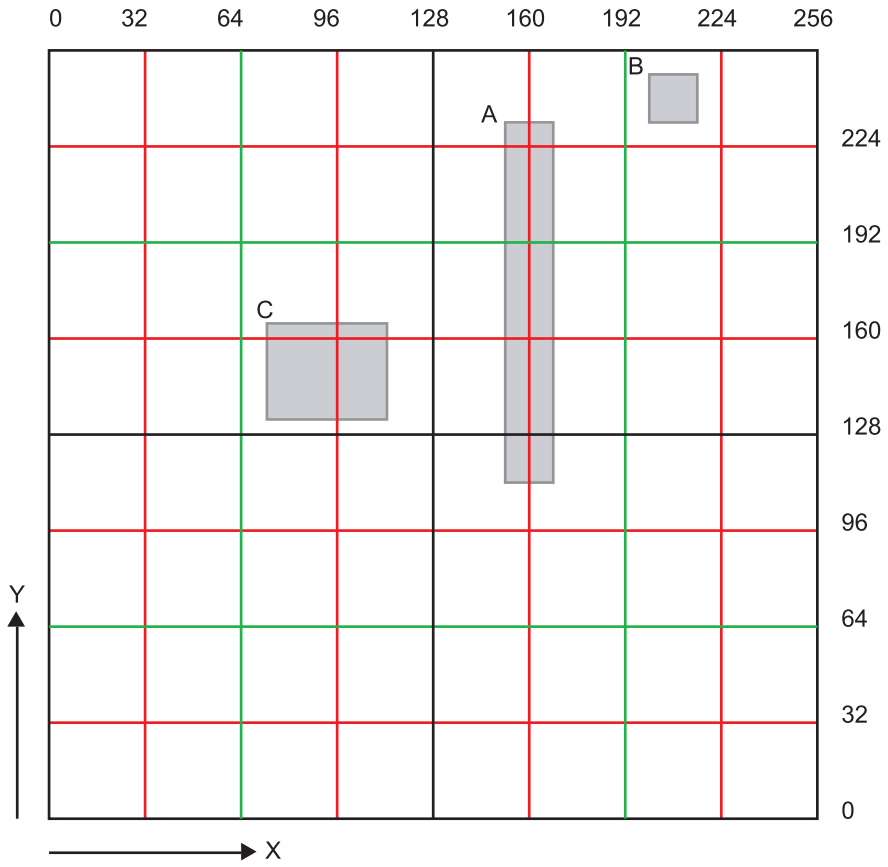
Předpokladem je, že maximální rozměr stromu bude odpovídat druhé mocnině čísla dvě. V této implementaci byl zvolen rozměr stromu 256. Kořen stromu má tedy rozměry 256 x 256. Další levely stromu jsou pak tvořeny půlením intervalu [0-255]. Například uzly v prvním levelu mají rozměry 128 x 128. Pro vložení entity do QuadTree stačí pouze souřadnice ve 2D, tedy z výškou dané entity se v tuto chvíli nepočítá. Ty jsou získány z AABB dané entity a následně přepočítány do souřadnicového systému stromu. Souřadnice tedy musí náležet intervalu [0-255].

Algoritmus vkládání začíná výpočtem cílového levelu, do kterého bude entita vložena. Umístění do cílového levelu je závislé na tom, která půlící hrana stromu protíná vkládanou entitu. Entita je ve stromu umístěna tak, aby žádnou s těchto půlících hran neprotínala. Umístění entit v jednotlivých levelech je patrné z obr. 18. Například objekt A z obr. 18 musí být umístěn v nejvyšším nultém levelu, protože je protnut dělící hranou prvního levelu 128. Pokud jsou tedy půlící hrany násobky druhé mocniny čísla dvě, je výpočet levelu realizován operací XOR jeho souřadnic. Ve výsledku XOR pak index nejvyššího bitu odečtený od maximálního možného levelu stromu, který je v tomto případě 7, představuje onen hledaný level. Výpočet se provede zvlášť pro x-ové a zvlášť pro y-ové souřadnice. Za cílový level je pak zvolen ten vyšší.

Výpis 1: Výpočet cílového levelu na základě souřadnic entity

```
unsigned short level = x0 ^ x1;
__asm mov   ax, 0x0
__asm bsr   ax, level
__asm mov   level, ax
level = 7 - level;
```

Zjištění nejvyššího nastaveného bitu ve výsledku XOR není zrovna efektivní implementovat v C++ v cyklu pomoci bitových operací. Mnohem efektivnější je využití x86 instrukce BSR (Bit Scan Reverse). Ukázka implementace je uvedena ve výpise 1. Proměnná *level* představuje na vstupu výsledek operace XOR dvou souřadnic. Na konci, po zjištění indexu nejvyššího nastaveného bitu a odečtení od maximálního možného levelu, cílový level.



Obrázek 18: Ukazka struktury QuadTree o hloubce čtyř levelů.

Na základě zjištěného cílového levelu a souřadnic vkládané entity zjistíme přesné umístění entity ve stromu v daném levelu. Tedy souřadnice daného uzly ve stromu. Pokud se má například objekt nacházet v prvním levelu a známe souřadnice tohoto objektu, pak tyto souřadnice musejí být transformovány do intervalu [0,1]. Toho se dá docílit pomoci operace *bitového posunu vpravo*. Na obr. 17 vezměme například objekt C, ten má souřadnice [72,165] a nachází se v levelu 2. Pak souřadnice x a y cílového uzlu v levelu

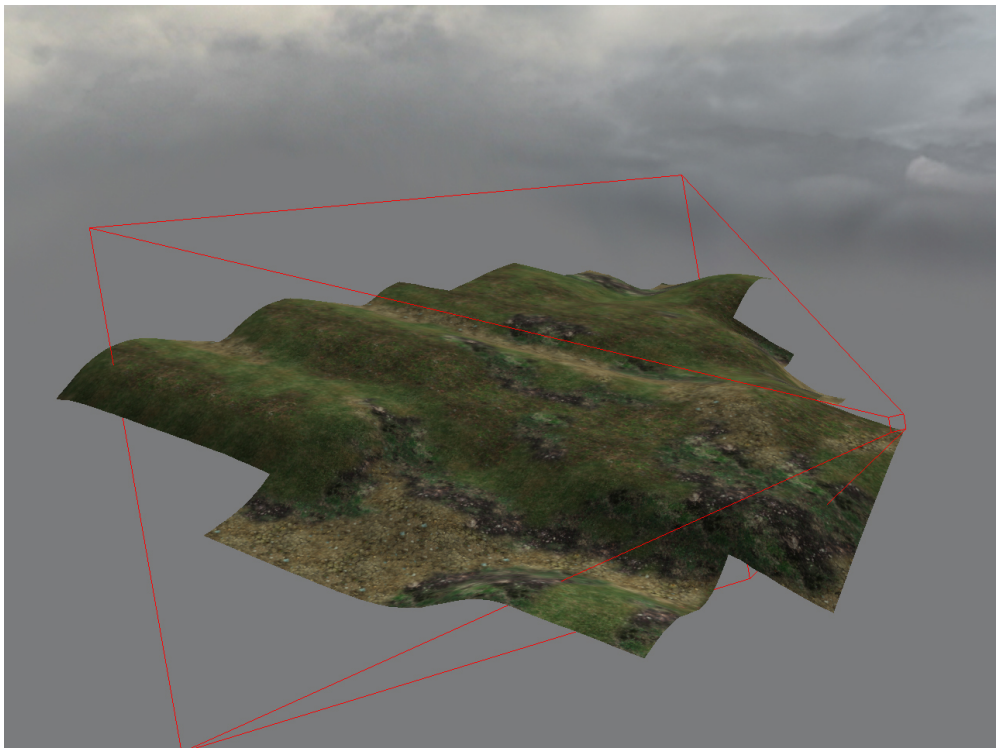
jsou vypočítány jako: `x = 72 >> 6` a `y = 165 >> 6`. Objekt C má tedy souřadnice uzlu [1,2] v levelu 2.

Na základě takto známých souřadnic je pak entita přímo vložena do stromu do konkrétního uzlu a levelu. Pro tento účel jsou také odkazy na jednotlivé uzly stromu udržovány v setřízeném poli. Vkládání do stromu je pak provedeno přímým přístupem do tohoto pole.

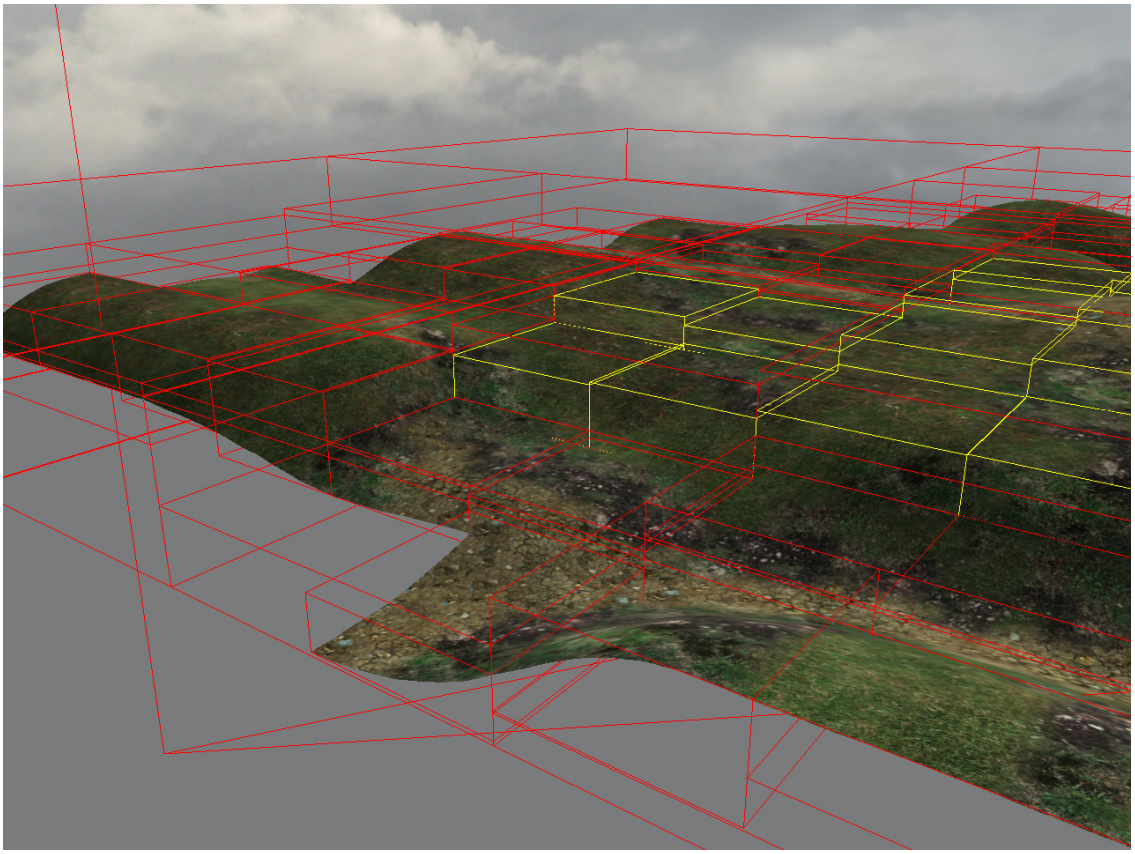
8.3 Výběr viditelných uzlů

Takto sestavený strom popisuje umístění všech entit ve scéně a umožňuje efektivně vybírat viditelné části scény. Při výběru viditelných sektorů je využit hraniční box (AABB), definující objem uzlu ve scéně, uchovaný v každém uzlu stromu, viz obr. 20. Strom je rekurzivně procházen od kořene dolů a na základě výpočtu kolize pohledového kužele a objemu uzlu je určeno, zda je daný uzel viditelný. Pokud uzel viditelný není, jsou jeho potomci již z dalšího zpracování vyloučeni. Pokud právě testovaný uzel viditelný je, pokračuje se směrem dolů a testují se potomci tohoto uzlu. Renderovatelné entity nacházející se ve viditelných uzlech jsou během procházení umísťovány do RenderQueue.

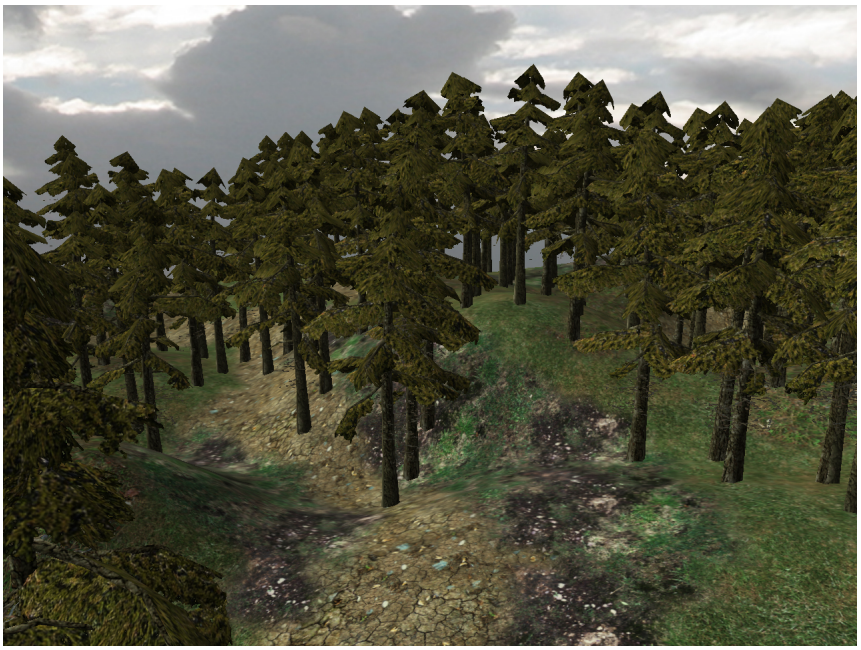
Na obr. 19 je screenshot z výsledné aplikace. Jedná se o ukázku vykreslováni terénu reprezentovaného strukturou QuadTree. Vykreslena je jen ta část terénu, která je viditelná kamerou. Na obrázcích 21 a 22 je pak ukázka z kompletní scény.



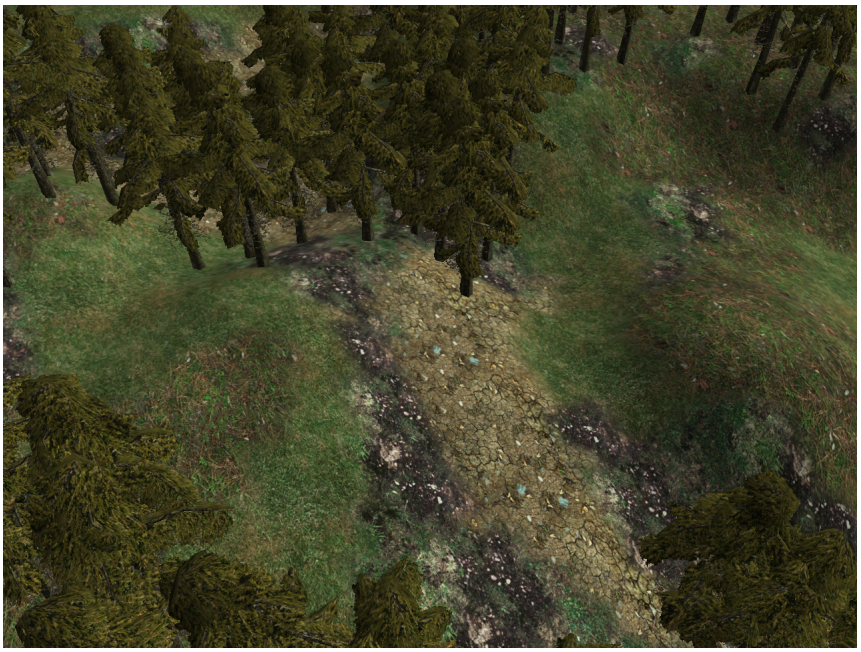
Obrázek 19: Vykreslovány jsou jen ty sektory terénu nacházející se v pohledovém objemu.



Obrázek 20: Vizualizace uzlů QuadTree dělících prostor scény.



Obrázek 21: Screenshot z výsledné aplikace.



Obrázek 22: Screenshot z výsledné aplikace.

9 Závěr

Cílem této práce není překonání kvalit dnes dostupných enginů, jak komerčních tak freewarových, ale spíše položit základ pro vytvoření vlastního jednoduchého enginu. Ten se stane následně základem vlastní počítačové hry *Heroes*. Implementace vlastního herního enginu dává možnost hlouběji proniknout do této né zrovna jednoduché problematiky. Základ enginu prezentovaný v této práci tvoří jen zázemí pro budoucí rozvoj tohoto projektu. V budoucnu bude kladen větší důraz na optimalizaci, zprávu scény a celkově větší ucelení enginu. Výrazně bude do enginu začleněna umělá inteligence, bez které se již žádná dnešní hra neobejde. Ta se z nemalé části podílí na hratelnosti a celkovém dojmu hry. I přes to, že dobrou hru dělá hlavně její hratelnost a atmosféra, tak zaostávat nebude ani grafická stránka. Navození správné atmosféry ve hře závisí koneckonců z větší části na dobrém grafickém zpracování. Možnosti dnešních grafických karet se stále rychleji vyvíjí, narůstá jak výkonnost tak i technické možnosti zobrazování 3D grafiky. Vytvoření vlastního enginu tak dává prostor pro vyzkoušení a nastudování těchto nových technologií a principů počítačové grafiky, které dnešní grafické karty umožňují. S využitím těchto možností dnešních grafických karet, bude tedy engine výrazně rozšířen také po grafické stránce.

Janošek Lumír

10 Reference

[1] Morgan Kaufmann Publishers Inc., *3D game engine design: a practical approach to real-time computer graphics*
San Francisco, CA, USA, 2000, ISBN 1-55860-593-2

[2] Christopher Tremblay, *Mathematics for Game Developers*
2004, ISBN-13 978-1592000388

[3] © 2010 Microsoft Corporation, *DirectX Developer Center*
http://msdn.microsoft.com/en-us/directx/default.aspx

[4] © 2010 Microsoft Corporation, *Windows DirectX Graphics Documentation*

[5] DeLoura Mark, *Game Programming Gems 2*
Rockland, MA, USA, 2001, ISBN-1584500549

[6] OGRE Team, *Stránky enginu OGRE*
http://www.ogre3d.org/

[7] Irrlicht Engine team, *Stránky Irrlicht Engine*
http://irrlicht.sourceforge.net/

[8] Crystal Space team, *Stránky Crystal Space SDK*
http://www.crystalspace3d.org/main/Main_Page

[9] Valve, *Stránky herního enginu Source*
http://source.valvesoftware.com/

[10] id Software, *Stránky herního enginu id Tech 4*
http://www.idsoftware.com/business/idtech4/

[11] Crytek GmbH, *Stránky herního enginu CryENGINE®3*
http://www.crytek.com/technology/cryengine-3/specifications/

A Obsah doprovodného CD

- Text této práce ve formátu pdf.
- Zdrojové kódy programu.
- Zkompilovaný a spustitelný program.
- Zkompilovaný program se zapnutou vizualizací QuadTree.
- Textury a modely využívané v aplikaci.
- Microsoft DirectX Redistributable.
- Microsoft Visual C++ 2010 Runtime Libraries.
- Screenshoty z aplikace.